AD-761 963

UNIDEC ASSEMBLER

C. Stephen Carr

Utah University

Prepared for:

Advanced Research Projects Agency

6 June 1968

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

AD 761963

Technical Report 4-6

C. Stephen Carr

UNIDEC ASSEMBLER

June 6, 1968

Computer Science

Information Processing Systems

University of Utah

Salt Lake City, Utah

JUN 26 1973

B

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U S Department of Commerce
Springfield VA 22151

Advanced Research Projects Agency · Department of Defense · ARPA Order 829

Program Code Number 6030

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

The difficulties in using the Project's PDP-8 computer are primarily due to its inadequate input/output facilities. Assemblies with paper tape and Macro-8 (manufacturer supplied) nominally require an hour to perform. As an alternative, the Unidec assembler runs on the Univac 1108 and passes assembled PDP-8 code over the electronic link between the two machines. The source statements are punched on cards for input into the 1108 in a format nearly identical to that of Macro-8. A printed listing and the object code are produced as fast as the cards can be read.

Of course, most users will prefer to write all programs for the 1108 and access the display facilities through the Graphics Monitor*. Indeed, this is why additional input/output facilities have not been purchased for the small PDP machine. However, Unidec meets the needs of those who require special PDP-8 programs.

---

* U.S. - Graphics System, Carr, C.S., Copeland, O.E., Information Processing Systems, Computer Science, University of Utah, Technical Report 4-1.

The syntax and operation of Unidec differ sufficiently from that of Macro-8 to warrant a separate user manual. However, the systems are sufficiently similar that excerpts from the DEC manual (Macro-8 Programming Manual-8-8-S, Digital Equipment Corp., Maynard, Mass.) have been copied unaltered. The implementation on the 1108 is entirely new.

F U N D A M E N T A L S

Characters

Programs in the Unidec language use the following characters:

Letters: A B C D . . . X Y Z

Digits: 1 2 3 4 5 6 7 8 9 0

Punctuation Characters:

Character		Use
␣	space	Combine symbols or numbers
+	plus	Combine symbols or numbers (add)
-	minus	Combine symbols or numbers (subtract)
!	exclamation point	Combine symbols or numbers (OR)
	end of card	Terminate line
,	comma	Assign symbolic address
=	equals	Define parameters
;	semicolon	Terminate coding line
$	dollar sign	End of pass
*	asterisk	Set location counter
.	point	Has value equal to current location counter

/	slash	Indicates start of a comment
&	ampersand	Combine symbols or numbers (and)
"	quote	Generate ASCII constant
()	parentheses	Define literal on current page
[]	brackets	Define page 0 literal
<>	angle brackets	Define a macro

All other characters are illegal when not in a comment or a TEXT field, and cause the error message E to be printed.

Source statements are entirely field free:

```
GO,  TAD TOTAL/MAIN LOOP/
```

but it is easier to read if spaces are inserted:

```
GO,  TAD TOTAL    /MAIN LOOP/
```

Either ; (semicolon) or the end of the card (column 72) terminates the line. The semicolon is considered identical to the end of a card except that it will not terminate a comment. Example:

```
TAD A                          /THIS IS A COMMENT ;  TAD
```

The entire expression beyond the slash is considered a comment.

Use of the semicolon allows several lines of coding to be on a single line: For example, a sequence of instructions to rotate the C(AC) and C(L) six places to the right might look like:

```
...
RTR
RTR
RTR
...
```

The above sequence may be rewritten as:

RTR; RTR; RTR

This type of format is particularly useful when the instructions work together conceptually.

A neat printout (or program listing, as it is usually called) makes subsequent editing, debugging, and interpretation much easier than if the coding were laid out in haphazard fashion.

Elements

Any group of letters, digits, and punctuation characters which represents binary values less than 2^{12} is an element.

Integers

Any sequence of numbers delimited by punctuation characters forms a number. Example:

1

12

4372

The radix control pseudo-instructions set the radix to be used in number interpretation. The pseudo-instruction DECIMAL indicates that all numbers are to be interpreted as decimal until the next occurrence of the pseudo-instruction OCTAL and conversely.

Symbols

A symbol is a string of one or more alphanumeric characters delimited by a punctuation character. Symbols are composed according to the following rules:

1. The characters must be either alphabetic (A-Z) or numeric (0-9).
2. The first character must be alphabetic.
3. Only the first six characters of any symbol are meaningful; the remainder, if any, are ignored.
4. A symbol is terminated by any nonalphanumeric character.

Unidec's permanent symbol table defines symbols for PDP-8 operation codes, operates commands, and many IOT commands (see Appendix A for a complete list). These may be used without prior definition by the user. Example:

JMS is a symbol whose value of 4000 is taken from the operation code definitions.

A is a user-created symbol. When used as a symbolic address tag, its value is the address of the instruction it tags. This value is assigned by the Assembler.

Note that because of rule 3, a symbol such as INTEGER, for instance, would be interpreted as INTEGE since the seventh letter is ignored. If symbols of more than six characters are used, the programmer should be careful to avoid the error of defining two apparently different symbols whose first six characters are, in fact, identical. For example, the two symbols GEORGE1 and GEORGE2 differ only in the seventh character and would be treated as GEORGE.

It is not necessary to define a symbol before it is used in an expression. They must be defined before the end of PASS1, however. Thus, one may refer to a number of registers by their address tags, and then define the symbols later.

Parameter Assignments

A symbol may be assigned a value by means of a parameter assignment statement which looks like an algebraic statement. The single symbol to the left of the equal sign is assigned the value of the expression on the right. Examples:

```
A = 6

EXIT = JMP  1 0

C = A + B
```

All symbols to the right of an "=" sign must be already defined. The symbol to the left of the "=" sign and its associated value is stored in the Assembler's symbol table.

The use of the "=" does not increment the current location counter. It is, rather, an instruction to the Assembler itself rather than a part of the output binary. The equal sign may be used to redefine a symbol.

Symbol Definition

A symbol may be defined by the user in one of three ways:

1. By use of a parameter assignment. Example:

```
DISMIS = JMP RESTOR
```

2. As a macro name. Example:

```
DEFINE LOAD A

<CLA
 TAD A>
```

3. By use of the comma. When a symbol is terminated by a comma, it is assigned a value equal to the current location counter.

Example:

```
     *100                          /SET CLC TO 100
TAG,     CLA
         JMP A
B,       0
A,       DCA B
         ...
         ...
```

The symbol "TAG" is assigned a value of 0100, the symbol "B" a value of 0102, and the symbol "A" a value of 0103.

Expressions

All elements, i.e., symbols and numbers (exclusive of pseudo-instruction symbols, and macro names) may be combined with certain operators to form expressions. These operators are:

+	plus	This signifies 2's complement addition (modulo 4096_{10}).
-	minus	This signifies 2's complement subtraction (modulo 4096_{10}).
!	exclamation point	This signifies Boolean inclusive OR (union).
&	ampersand	This signifies Boolean AND (intersection).
␣	space	Space is interpreted in context. It may signify inclusive OR or act as a field delimiter.

Symbols and integers may be combined with any of the above operators. A symbolic expression is evaluated from left to right; no grouping of terms is permitted. Example:

	A	B	A+B	A-B	A!B	A&B
Value	0002	0003	0005	7777	0003	0002
Value	0007	0005	0014	0002	0007	0005
Value	0700	0007	0707	0671	0707	0000

Unidec makes a distinction between the types of symbols it is processing. These types are 1) permanent symbols, 2) user defined symbols, and 3) macro names. The character "space" is interpreted written in the context of the expression. If a space is used to delimit two or more permanent symbols, space signifies inclusive OR. Example:

```
CLA     is a permanent symbol whose value is 7200.
CMA     is a permanent symbol whose value is 7040.
```

The expression:

CLA␣CMA has a value of 7240.

If the symbol following the space is a user defined symbol, space acts as an address field delimiter. Example:

```
  *2117
A,    CLA
      ...
      ...
      JMP␣A
```

"A" is a user defined symbol whose value is 2117. The expression JMP␣A is evaluated as follows:

The seven address bits of A are taken, i.e.:

```
A 010 00 1 001 111
        1 001 111
```

The remaining five bits of A are tested to see if they are 0's (page 0 reference); if they are not, the current page bit is set.

```
000 011 001 111
```

The operation code is ORed into the expression:

```
      JMP 101 000 000 000
Address A 000 011 001 111
JMP     A 101 011 001 111
```

or, written more concisely:

```
5317.
```

In addition to the above outlined tests, the page bits of the address field are compared with the page bits of the current location counter. If the page bits of the address field are nonzero and do not equal the page bits of the current location counter, an out-of-page reference is being attempted. If the reference is to an address not on the page where the instruction will be located, the Assembler will set the indirect bit (bit 3) and an indirect address linkage will be generated on the current memory page. If the out-of-page reference is already an indirect one, the error diagnostic I (Illegal Indirect) will be printed. In the case of several out-of-page references to the same address, the link will be generated only once.

Example:

```
    *2117
A,      CLA
        .
        .
        .
*2600
        JMP␣A
```

The space preceding the user defined symbol "A" acts as an address field delimiter. The Assembler will recognize that the register tagged "A" is not on the current page (in this case 2600-2777) and will generate a link to it as follows:

in location 2600 the Assembler will place the word

5777 which is JMP 1 2777

in address 2777 (the last location on the current page), the word 2117 (the actual address of "A" will be placed.

The address field of a storage reference instruction may be any valid expression. Example:

```
A=270
*200
TAD A-20
```

would produce, in location 200, the word

001 010 101 000 or 1250 (TAD 250)

Although the Assembler will recognize and generate an indirect address linkage when necessary, the programmer may indicate an explicit indirect address by using the special symbol "1". This must be between the operation code and the address field. The Assembler cannot generate a link for an instruction that is already specified as being an indirect reference. In this case, the Assembler will print I (Illegal Indirect).

Current Address Indicator

The single character period (.) has, at all times, a value equal to the value of the current location counter. It may be used as any integer or symbol (except to the left of an equal sign). Example:

```
*200
JMP .+2
```

Is equivalent to JMP 202.

```
*300
.+2400
```

would produce, in register 0300, the quantity 2700.

```
    *2200
CALL=JMS 1.
    0027
```

Since the second line, CALL=JMP 1., does not increment the current location counter, 0027 would be placed in register 2200 and CALL would be placed in the symbol table with an associated value of

100 110 000 000 or 4600.

Origin Setting

The origin (current location counter) is reset by use of the special character asterisk (*). The current location counter is set to the value of the expression following the "*". The origin is initially set to 0200. All symbols to the right of "*" must already have been defined. Example:

If D has the value 250

then

*D+10 will set the location counter to 0260.

To simplify page handling, the pseudo-instruction PAGE may be used. When "PAGE" is encountered, the origin is reset to the first location of the next page. A page number may be specified by a legal expression following the page pseudo-instruction. Example:

```
*270
...
...
```

at this point, either

```
*400
PAGE
```

or

```
PAGE 2
```

will reset the origin to 0400.

Literals

Since the symbolic expressions which appear in the address part of an instruction usually refer to the locations of registers containing the quantities being operated upon, the programmer must explicitly reserve the registers holding his constants or use a literal. Suppose, for example, that the programmer has an index which is incremented by two. One way of coding this operation would be as follows:

```
    ...
    CLA
    TAD INDEX
    TAD C2
    DCA INDEX
    ...
    ...
C2,2
```

Using a literal, this would become:

```
    ...
    CLA
    TAD INDEX
    TAD (2)
    DCA INDEX
    ...
```

The left parenthesis is a signal to the Assembler that the expression following is to be evaluated and assigned a register in the constants table of the current page. This is the same table in which the indirect address linkages are stored. In the above example, the quantity 2 is stored in a register in a list beginning at the top of the storage page (page address 177), and the instruction in which it appears is encoded with an address referring to that address. A literal is assigned to storage the first time it is encountered; subsequent references will be to the same register.

If the programmer wishes to assign literals to page 0 rather than the current page, he may use square brackets, "[" and "]" in place of the parentheses. However, in both cases, the right of closing member may be omitted. The following examples are acceptable:

```
TAD (777
AND [JMP
```

Note that in the second example, the instruction AND [JMP has the same effect as AND [5000.

Literals may be nested. For example:

```
*200
TAD  (TAD   (30
```

will generate

```
0200      1376
...       ...
0376      1377
0377      0030
```

This type of nesting may be carried to as many levels as desired. Literals are stored on each page starting at page address 177 and extending toward page address 0. (Only 127_{10} or 177_8 literals may be placed on page 0). If a literal has been generated for a nonzero page and then the origin is set to another page, the current page literal buffer is emptied onto the old page where they are referenced. If the origin is then reset to the previously used page, the same literal will be generated if used again.

Single Character Text Facility

If a single character is preceded by a double quote, the 8-bit value of the ASCII code for the character is inserted instead of taking the letter as a symbol. Example:

```
CLA
TAD  ("A
...
```

will place the constant 0301 in the accumulator.

P S E U D O - I N S T R U C T I O N S

The pseudo-instructions are directions to the Assembler to perform certain tasks or to interpret subsequent coding in a certain way. By themselves, pseudo-instructions do not generate coding or (in general) effect the current location counter.

Current Location Counter

PAGE — This pseudo-instruction is used to set the current location counter.

PAGE n — This will reset the current location counter (CLC) to the first address of page n, where n is an integer, a previously defined symbol, or a symbolic expression. Examples:

PAGE 2 will set the CLC to 0400

PAGE 6 will set the CLC to 1400

PAGE — When used without an argument, PAGE will reset the CLC to the first location on the next succeeding page. Thus, if a program is being assembled into page 1 and the programmer wishes to begin the next segment on page 2, he need only insert the pseudo-instruction PAGE, as follows:

```
        JMP     .-7

    PAGE

        CLA
```

The current location counter may be explicitly set by use of the asterisk.

Radix Control

Normally, all integers used in a program are taken as octal numbers. If, however, the programmer wishes to have certain numbers treated as decimal, he may use the pseudo-instructions:

DECIMAL — When this pseudo-instruction occurs, all integers encountered in subsequent coding will be taken as decimal until the occurrence of the pseudo-instruction

OCTAL — which will reset the radix to its original (octal) base.

Text Facility

There is a text facility for single characters and text strings. The single character mode (double quote) has been described previously.

A string of text may be entered by giving the pseudo-instruction TEXT followed by a space, a delimiting character, a string of text, and repeating the same delimiting character. Example:

```
TEXT    ATEXTA
```

The character codes are stored two to a register in ASCII code that has been trimmed to six bits. Following the last character, a 6-bit zero is inserted as a stop code. The above statement would produce

```
2405

3024
0000
```

```
TEXT␣   /BOB/
```

would produce

```
0217
0200
```

Note that while the TEXT pseudo-instruction causes characters to be stored in a trimmed code, the use of the single-character control code (") causes characters to be stored as a full 8-bit ASCII code.

End of Program

The special symbol "$" indicates the end of a program. When the Assembler encounters the "$", it terminates the PASS. Several programs may be assembled at one time. A dollar sign terminates each program.

M A C R O S

When writing a program, it often happens that certain coding sequences are used several times with just the arguments changed. If so, it is convenient if the entire sequence can be generated by a single statement. To do this, the coding sequence is defined with dummy arguments as a macro. A single statement referring to the macro by name, along with a list of real arguments, will generate the correct sequence in line with the rest of the coding. DEC's Macro-8 provides a weak macro facility which Unidec provides for compatibility.

The macro name must be defined before it is used. The macro is defined by means of the pseudo-instruction DEFINE followed by the macro's name and list of dummy arguments. For example:

A macro to move the contents of register A to refister B and also leave the result in the accumulator, would be coded as follows:

```
DEFINE␣MOVE␣DUMMY1␣DUMMY2
<CLA
TAD     DUMMY1
DCA     DUMMY2
TAD     DUMMY2>
```

The actual choice of symbols used as dummy arguments is arbitrary; however, they may not be defined or referenced prior to the macro definition.

The above definition of the macro MOVE is identical to the following:

```
DEFINE␣MOVE␣ARG1␣ARG2
<CLA:TAD   ARG1;   DCA ARG2;   TAD ARG2>
```

The actual definition of the macro is enclosed in angle brackets.

When a macro name is processed by the assembler, the real arguments will replace the dummy arguments. For example:

Assuming that the macro MOVE has been defined as above,

```
*400
A,0         0400      0000
B,-6        0401      7772
MOVE  A,B   0402      7200
$           0403      1200
            0404      3201
            0405      1201
```

NOTE: A macro need not have any arguments: For example, a sequence coding to rotate the C(AC) and C(L) six places to the left might be encoded as a macro by means of

```
DEFINE    ROTL 6
<RTL;RTL;RTL>
```

The macro definition can consist of any valid coding except for TEXT or " type statements.

Restrictions

1. Macros cannot be nested; i.e., another macro name or definition cannot appear in a macro definition and cannot be brought in as an argument at reference time.
2. TEXT or " type statements cannot appear in a macro definition.
3. Arguments cannot be:
   a. Macro name
   b. TEXT pseudo-instruction or " special character
4. The symbols used as dummy arguments must not have been previously defined or referenced.
5. A macro may not be redefined. Example:

```
DEFINE␣LOOP␣A␣B

<TAD   A
 DCA   B
 TAD   COUNT
 ISZ   B
 JMP   .-2>
```

The symbol "COUNT" is not a dummy argument but an actual symbol.

A macro is referenced by giving the macro name, a space, and then the list of real arguments, separated by commas. There must be at least as many arguments in the macro reference as in the corresponding macro definition. When a macro is referenced, its definition is found, expanded, and the real arguments replace the dummy arguments. The expanded macro is then processed in the normal fashion.

```
....
LOOP␣X,Y2
....
```

is equivalent to:

```
....
TAD     X
DCA     Y2
TAD     COUNT
ISZ     Y2
JMP     .-2
....
```

OPERATING INSTRUCTIONS

1. Set up an 1108 deck.

   ∇ RUN __,_____,__,____

   ∇ G=$PDPS$

   ∇ XQT CUR

   IN G

   ∇ XQT UNIDEC

PDP-8 program

   $ ←(end of assembly indicator)

next PDP-8 program

   $

2. The assembler uses the ARPAIO program. Therefore, ready the PDP as explained elsewhere.
3. Run the 1108 deck into the 1004.
4. The assembled program goes directly into the PDP core storage. A listing is produced on the 1004.

NOTE: The following PDP locations are used by the link program:

1, 3,
10, 11
20, 21
7000 - 7777

All other locations are available. Unidec assumes a starting location of 200_8 if no other origin information is given.

A P P E N D I X A

Unidec Permanent Symbol Table

AND	0000	SZA	7440	TSF	6041
TAD	1000	SNA	7450	TCF	6042
ISZ	2000	SNL	7420	TPC	6044
DCA	3000	SZL	7430	TLS	6046
JMS	4000	SKP	7410	NOP	7000
JMP	5000	OSR	7404	CLA	7200
IOT	6000	HLT	7402	STA	7240
OPR	7000	CIA	7041	SPA	7510
CLA	7200	LAS	7604	I	0400
CLL	7100	STO	7240	VEC	7000
CMA	7040	STL	7120	JUMP	7400
CML	7020	GLK	7204	DOT	7410
RAR	7010	ION	6001	DASH	7420
RTR	7012	IOF	6002	LINE	7430
RAL	7004	KSF	6031	FRAM	7440
RTL	7006	KCC	6032	SYMB	7200
IAC	7001	KRS	6034	FIN	5407
SMA	7500	KRB	6036		

APPENDIX B

Alphanumeric Character Codes

Symbol ASCII Fieldata Trimmed-ASCII Card

Symbol	ASCII	Fieldata	Trimmed-ASCII	Card
Ø (zero)	60	60	20	0
1	61	61	21	1
2	62	62	22	2
3	63	63	23	3
4	64	64	24	4
5	65	65	25	5
6	66	66	26	6
7	67	67	27	7
8	70	70	30	8
9	71	71	31	9
A	101	06	41	12-1
B	102	07	42	12-2
C	103	10	43	12-3
D	104	11	44	12-4
E	105	12	45	12-5
F	106	13	46	12-6
G	107	14	47	12-7
H	110	15	50	12-8
I	111	16	51	12-9
J	112	17	52	11-1
K	113	20	53	11-2
L	114	21	54	11-3
M	115	22	55	11-4
N	116	23	56	11-5
O	117	24	57	11-6
P	120	25	60	11-7
Q	121	26	61	11-8
R	122	26	62	11-9
S	123	30	63	0-2
T	124	31	64	0-3
U	125	32	65	0-4
V	126	33	66	0-5
W	127	34	67	0-6
X	130	35	70	0-7
Y	131	36	71	0-8
Z	132	36	72	0-9

Symbol	ASCII	Fieldata	Trimmed-ASCII	Card
a	141	-	-	-
b	142	-	-	-
c	143	-	-	-
d	144	-	-	-
e	145	-	-	-
f	146	-	-	-
g	147	-	-	-
h	150	-	-	-
i	151	-	-	-
j	152	-	-	-
k	153	-	-	-
l	154	-	-	-
m	155	-	-	-
n	156	-	-	-
o	157	-	-	-
p	160	-	-	-
q	161	-	-	-
r	162	-	-	-
s	163	-	-	-
t	164	-	-	-
u	165	-	-	-
v	166	-	-	-
w	167	-	-	-
x	170	-	-	-
y	171	-	-	-
z	172	-	-	-